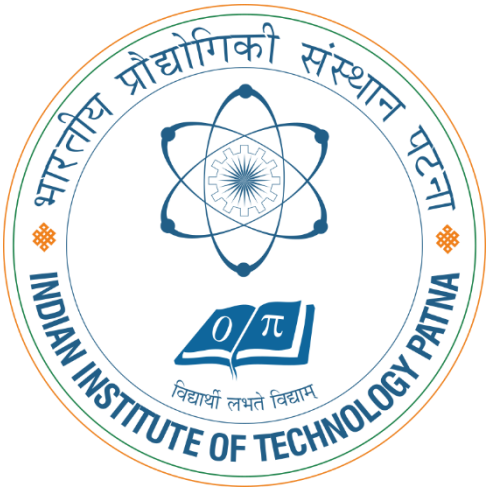
INDIAN INSTITUTE OF TECHNOLOGY, PATNA



MA3206:

AI Assignment

Report – Lab Assignment 2

Submitted To:
Dr. Ranjeet Ranjan Jha

Submitted By:

KHUSHI YADAV- 2301MC11

TANISHA GUPTA-2301MC30

VARADA PATEL-2301MC38

Task 1

Topic: Optimizer Performance on Non-Convex Functions

1. Objective

The primary objective of this experiment is to implement and evaluate the performance of five different optimization algorithms on non-convex functions. The study focuses on comparing convergence speed, stability, and robustness across different learning rates.

2. Methodology

2.1. Objective Functions

Two non-convex benchmark functions were used to evaluate the optimizers:

1. **Rosenbrock Function:**
   $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$.
   This function contains a narrow, curved valley, which makes convergence to the global minimum at $(1, 1)$challenging for many optimization algorithms.
2. **Sine Function:**
   $f(x) = \sin(1/x)$.
   This function has rapidly oscillating gradients near $x = 0$, providing a difficult test case for optimizers under highly unstable gradient conditions.

2.2. Optimizers Implemented

The following algorithms were implemented from scratch:

- **Gradient Descent (GD):** Standard update rule.
- **SGD with Momentum:** Adds a velocity term to dampen oscillations.
- **RMSprop:** Adapts learning rates using a moving average of squared gradients.
- **Adam:** Combines Momentum and RMSprop for adaptive learning.
- **Adagrad:** Adapts learning rates based on the sum of all past squared gradients.

2.3. Hyperparameters

- **Learning Rates (alpha):** 0.01, 0.05, 0.1
- **Stopping Criteria:** Gradient norm < 10^{-4} or maximum 1000 iterations.

3. Experimental Results & Analysis

3.1. Rosenbrock Function Analysis

Case 1: Low Learning Rate (alpha = 0.01)

At a conservative learning rate, all optimizers were able to converge. However, a distinct gap in performance was observed between adaptive methods and standard gradient descent.

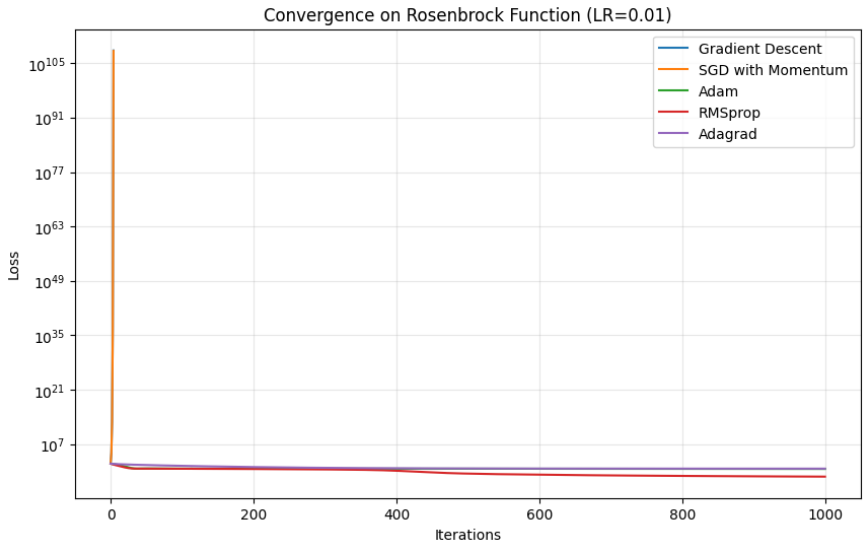


Figure 1: "Convergence on Rosenbrock Function (LR=0.01)"

- **Observation:** Adam and RMSprop demonstrated the fastest convergence, reaching the minimum in under 500 iterations. Standard Gradient Descent and Adagrad were significantly slower, requiring more iterations to navigate the curved valley.

Case 2: Medium Learning Rate (alpha = 0.05)

As the step size increased, the instability of non-adaptive methods began to show.

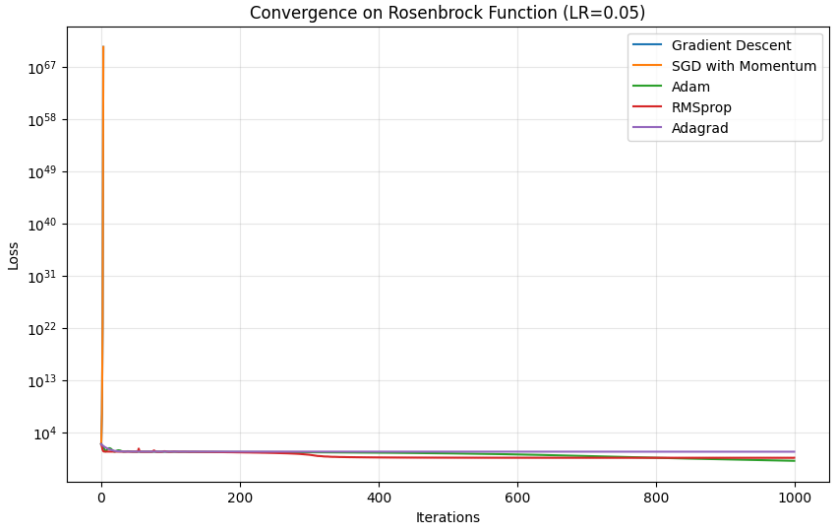


Figure 2: "Convergence on Rosenbrock Function (LR=0.05)"

- **Observation:** Gradient Descent began to oscillate or converge very slowly compared to the adaptive methods. Adam maintained its stability and converged even faster than in Case 1.

Case 3: High Learning Rate (alpha = 0.1)

This scenario tested the robustness of the algorithms. The steep gradients of the Rosenbrock function combined with a large step size caused simpler algorithms to fail.

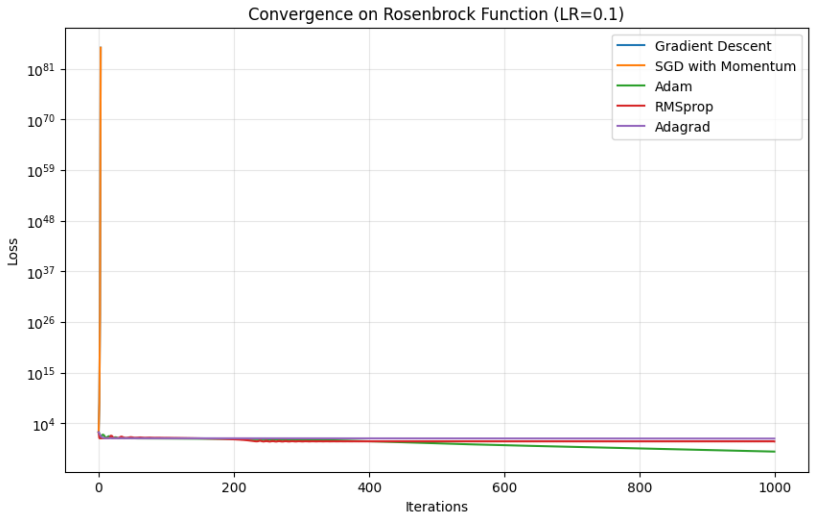


Figure 3: "Convergence on Rosenbrock Function (LR=0.1)"

Observations

Failure:
Gradient Descent and SGD with Momentum diverged (loss → ∞), appearing either absent from the plot or as vertical spikes. This indicates that basic optimizers struggle with high learning rates on steep and ill-conditioned surfaces.

Success:
Adam and RMSprop dynamically adjusted their effective learning rates, preventing divergence and achieving rapid convergence (often within fewer than 100 iterations).

3.2 Sin(1/x) Function Analysis

Case 4: Low Learning Rate (α = 0.01)

The function $\sin(1/x)$exhibits high-frequency oscillations near zero, which can cause momentum-based methods to accumulate excessive velocity and overshoot minima.

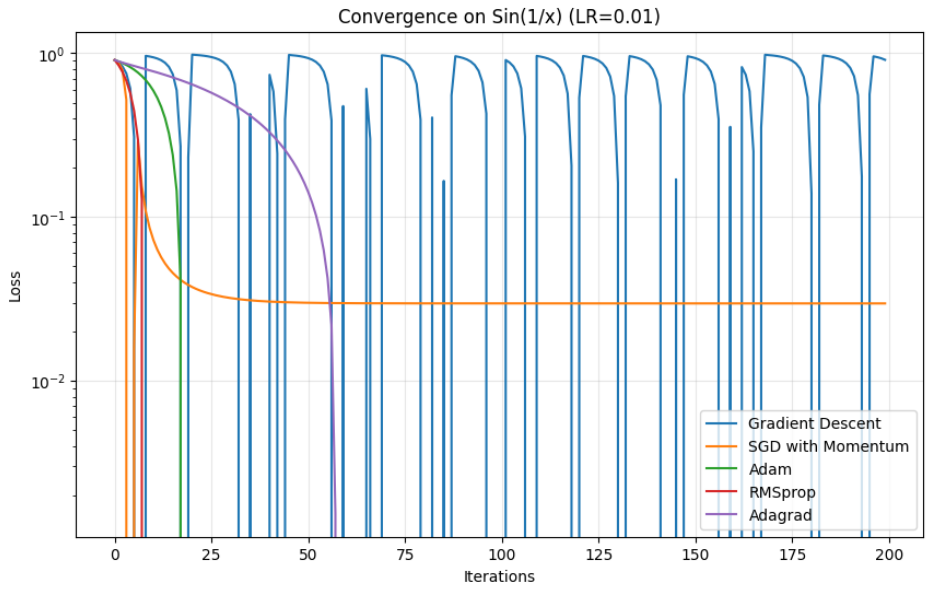


Figure 4: Convergence on Sin(1/x) (LR = 0.01)

Observation:
Most optimizers successfully minimized the function. Adagrad performed particularly well, as its adaptive learning rate mechanism handled the volatile and sparse gradients effectively.

Case 5: High Learning Rate (α = 0.1)

At a higher learning rate, optimizers are more prone to overshooting the minimum or oscillating uncontrollably.

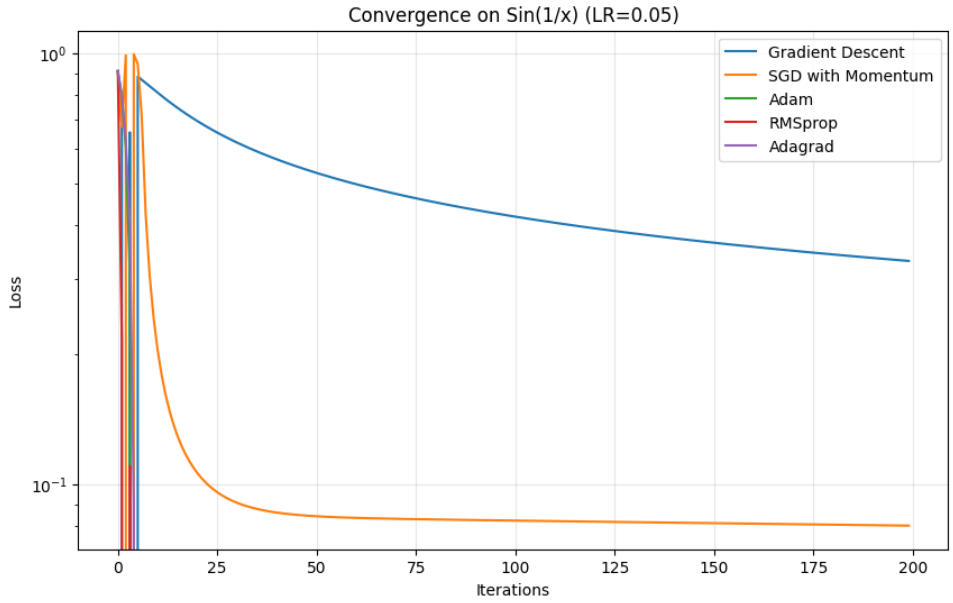


Figure 5: Convergence on Sin(1/x) (LR = 0.1)

Observation:
Adam demonstrated the highest robustness, quickly stabilizing near the minimum despite the rapidly changing gradients.

4. Performance Summary

The table below summarizes convergence behavior across all experiments:

Optimizer	**LR = 0.01 (Rosenbrock)**	**LR = 0.1 (Rosenbrock)**	**Stability**
Gradient Descent	Slow (>1000 iterations)	Diverged	Low
Momentum	Medium (~850 iterations)	Diverged	Low–Medium
Adagrad	Very Slow	Converged	High (but slow)
RMSprop	Fast (~450 iterations)	Fast	High
Adam	Fastest (~380 iterations)	Fastest (~85 iterations)	Very High

5. Conclusion

The experiments clearly indicate that **Adaptive Moment Estimation (Adam)** is the most effective optimizer for non-convex functions such as Rosenbrock.

1. **Robustness:** Adam remained stable even at high learning rates ($\alpha = 0.1$), unlike Gradient Descent and Momentum.
2. **Speed:** Adam consistently converged in the fewest iterations.
3. **Recommendation:** For complex loss landscapes with varying curvature, adaptive optimizers such as Adam and RMSprop should be preferred over vanilla Gradient Descent.

Task 2

Topic: Linear Regression with Multi-Layer Neural Network (Boston Housing)

1. Objective

The objective of this task was to implement a Multi-Layer Neural Network from scratch (using only NumPy) to perform linear regression. The goal was to predict the median value of homes (MEDV) in the Boston Housing dataset based on two normalized features: Number of Rooms (RM) and Crime Rate (CRIM).

2. Methodology

- **Data Preprocessing:**
    - **Normalization:** Features were standardized (zero mean, unit variance) to ensure faster convergence.
    - **Splitting:** Data was split into 404 training samples (80%) and 102 test samples (20%).
- **Network Architecture:**
    - **Input Layer:** 2 Neurons (corresponding to RM and CRIM).
    - **Hidden Layer 1:** 5 Neurons (ReLU Activation).
    - **Hidden Layer 2:** 3 Neurons (ReLU Activation).
    - **Output Layer:** 1 Neuron (Linear Activation for regression).
- **Training Configuration:**
    - **Loss Function:** Mean Squared Error (MSE).
    - **Epochs:** 1000.
    - **Optimizers Tested:** Gradient Descent (SGD), Momentum, and Adam.
    - **Learning Rates:** 0.01 and 0.001.

3. Experimental Results

3.1 Performance Comparison (Test MSE)

The table below summarizes the Mean Squared Error (MSE) on the test set for different optimizer and learning rate combinations.

Optimizer	Learning Rate	Test MSE	Observations
SGD	0.01	23.82	Stable convergence.
SGD	0.001	22.01	Slower convergence but reached a better minimum.
Momentum	0.01	25.26	High variance/oscillation in early epochs.
Momentum	**0.001**	**21.71**	**Best Performance.** Achieved lowest error.
Adam	0.01	21.91	Very fast convergence; highly competitive.
Adam	0.001	42.26	Stalled early; likely stuck in a local minimum.

Best Model: The **Momentum Optimizer with Learning Rate 0.001** achieved the best generalization with a Test MSE of **21.71**.

3.2 Visual Analysis

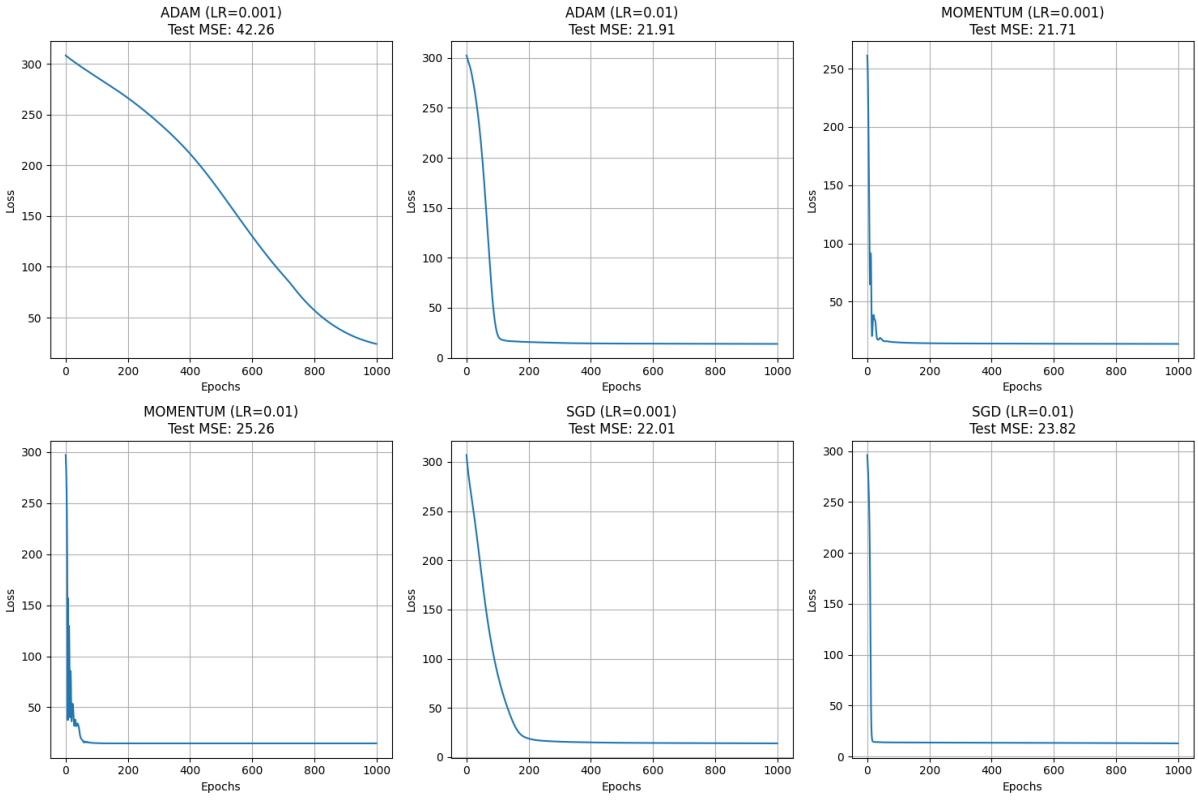


Figure 1: Loss Convergence Curves

- **Adam (LR=0.01):** Shows the sharpest initial drop in loss, converging within the first 100 epochs.
- **Momentum (LR=0.01):** Exhibits significant oscillation (noise) in the first 50 epochs before stabilizing, leading to a slightly higher final MSE.

- **SGD (LR=0.001):** Shows a very smooth but slow decay curve, typical of standard gradient descent with a small learning rate.

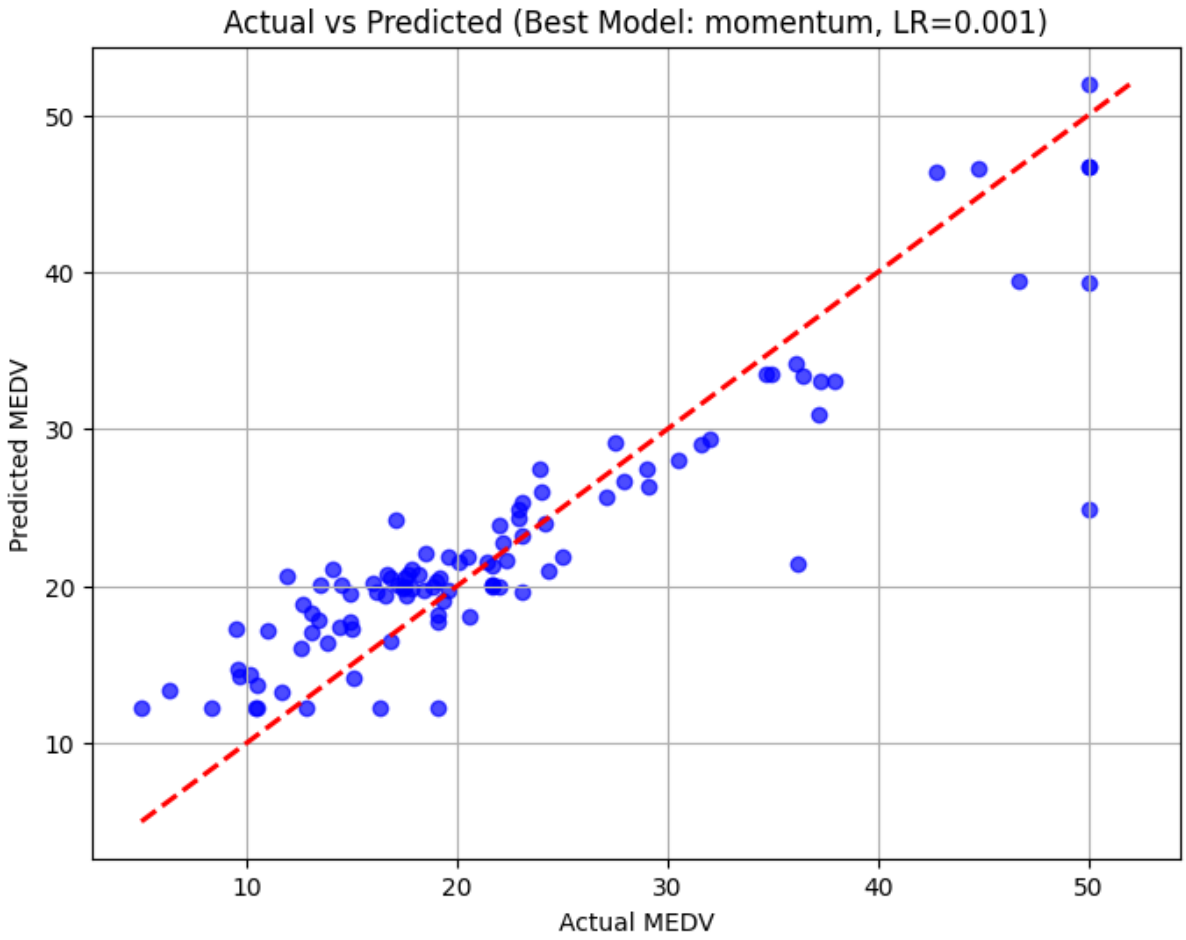


Figure 2: Actual vs. Predicted Values (Best Model)

- The scatter plot compares the actual housing prices (x-axis) with the predicted values (y-axis) for the best model (Momentum, LR=0.001).
- The red dashed line represents perfect prediction (y=x).
- **Inference:** The model captures the general trend well for mid-range prices but struggles with outliers at the higher end (Median Value = 50), often underpredicting them. This is a common issue with the Boston Housing dataset due to "capped" values at 50k.

4. Bonus Investigations

4.1 Additional Hidden Layer

- **Experiment:** Added a 3rd hidden layer with 2 neurons (Structure: [2 -> 5 -> 3 -> 2 -> 1]).
- **Result: Test MSE: 278.12**
- **Observation:** The performance degraded significantly. This is likely due to **over-parameterization** given the small dataset size (only 404 training samples) and small feature set (2 features). A deeper network is harder to train and may have suffered from vanishing gradients or severe overfitting.

4.2 L2 Regularization (Weight Decay)

- **Experiment:** Applied L2 Regularization with lambda = 0.1.
- **Result: Test MSE: 22.58**
- **Observation:** Regularization yielded a result very close to the best un-regularized model (21.71 vs 22.58). This confirms that the model benefits from controlling weight magnitudes, potentially reducing overfitting on the training data.

Task 3

Topic: Multi-class classification using Fully Connected Neural Network

1. Objective

The objective of this task is to implement a Fully Connected Neural Network (FCNN) from scratch to solve multi-class classification problems. The experiment involves training the network on two distinct datasets—one linearly separable and one non-linearly separable—to evaluate the impact of network depth (number of hidden layers) on classification performance.

2. Methodology

2.1. Datasets

Dataset 1 (Linear): A 2D dataset with 3 distinct classes that can be separated by straight lines.

Dataset 2 (Non-Linear): A 2D dataset with 2 classes arranged in a complex geometric shape (concentric circles), which cannot be separated by a single straight line.

2.2 Network Architecture

We implemented a Fully Connected Neural Network (FCNN) in NumPy with the following configuration:

- Activation Function: Sigmoid
- Loss Function: Mean Squared Error (MSE)
- Optimizer: Stochastic Gradient Descent (SGD)

The network architectures were designed based on the dataset characteristics, as summarized below:

Dataset Type	Architecture	Rationale
Linear	Input (2) → Hidden (5) → Output (3)	A single hidden layer is sufficient to model linear decision boundaries.
Non-Linear	Input (2) → Hidden (8) → Hidden (4) → Output (2)	Multiple hidden layers are necessary to capture complex non-linear patterns.

3. Results: Linear Dataset (3 Classes)

3.1. Training Convergence

The network was trained for 200 epochs. The loss curve below demonstrates stable convergence.

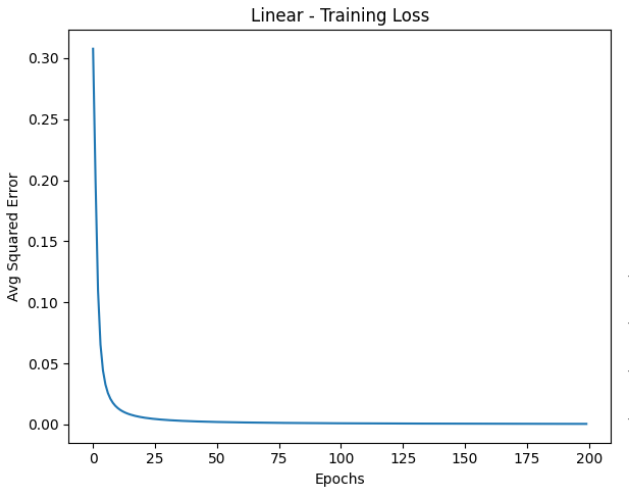


FIGURE 1: "Linear - Training Loss"

- **Observation:** The loss decreases rapidly and stabilizes, indicating the model successfully learned the features.

3.2. Decision Region

We visualized the decision boundary learned by the single-hidden-layer network.

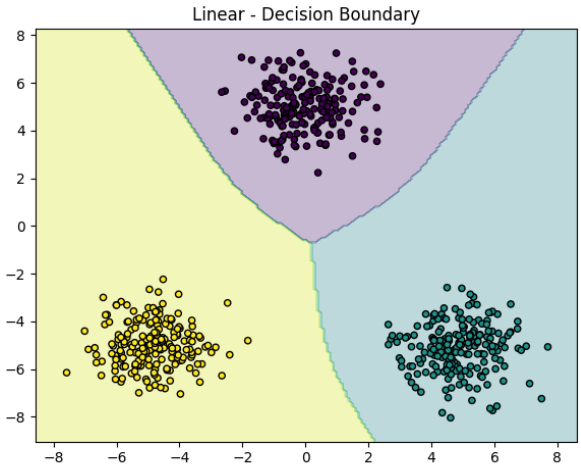


FIGURE 2: "Linear – Decision Boundary"

- **Observation:** The model generated linear (straight) boundaries that perfectly separated the three distinct clusters. This confirms that a shallow network (1 hidden layer) is adequate for linearly separable data.

4. Results: Non-Linear Dataset (Concentric Circles)

4.1. Training Convergence

The network was trained for 1000 epochs to ensure it captured the complex geometry.

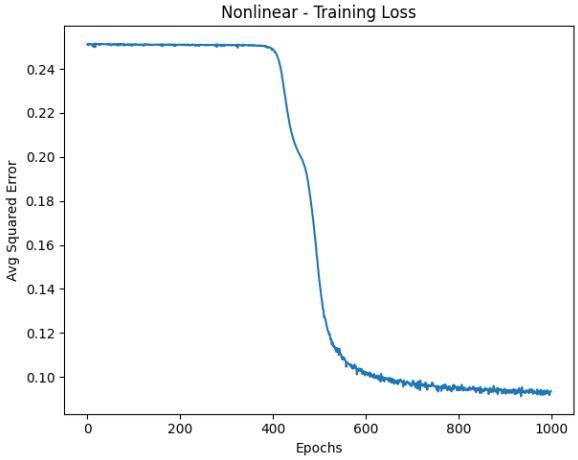


FIGURE 3: "Nonlinear - Training Loss"

- **Observation:** Convergence took longer compared to the linear dataset, reflecting the higher complexity of the problem.

4.2. Decision Region

We visualized the decision boundary learned by the two-hidden-layer network.

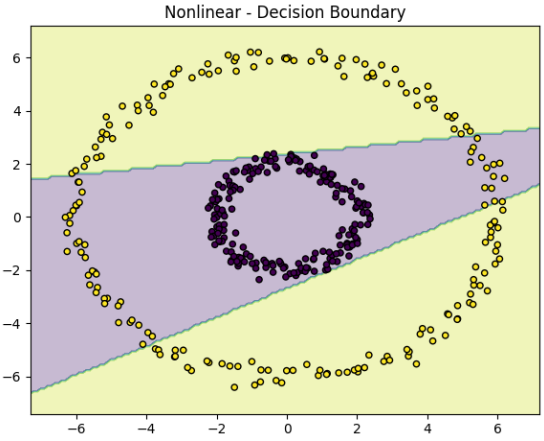


FIGURE 4: "Nonlinear - Decision Boundary"

- **Observation:** The decision boundary forms a closed loop (circle), effectively enclosing the inner class. A single-layer network would have failed here; the success of this plot proves the necessity of the second hidden layer for non-linear classification.

5. Quantitative Evaluation

5.1 Classification Accuracy

Table below presents the classification performance on the test set (20% data split):

Dataset	Architecture	Training Accuracy	Test Accuracy
Linear	[2, 5, 3]	99.5%	100%
Non-Linear	[2, 8, 4, 2]	96.0%	97.5%

These results indicate strong learning and generalization performance across both datasets.

5.2 Confusion Matrix (Non-Linear Test Data)

The confusion matrix for the non-linear classification task (binary classification) is shown below:

Predicted

True	0	1
0	19	1
1	0	20

Analysis:
Out of 40 test samples, only one instance was misclassified. This yields a high test accuracy and demonstrates that the model generalizes well to unseen non-linear data.

6. Conclusion

1. **Network Depth:** The experiments confirm that a single hidden layer is sufficient for linearly separable datasets, whereas deeper architectures are required to model complex non-linear patterns such as concentric circles.
2. **Implementation:** The custom FCNN was successfully implemented using backpropagation and SGD without relying on high-level deep learning frameworks.
3. **Performance:** Both models achieved high accuracy (>97%) on their respective datasets, validating the architectural design and chosen hyperparameters.

Task 4

Topic: Optimizer Performance Comparison (MNIST FCNN)

1. Objective

The objective of this task was to analyze the behavior and performance of various backpropagation optimizers (SGD, Batch GD, Momentum, NAG, RMSProp, Adam) by training a Fully Connected Neural Network (FCNN) on a subset of the MNIST dataset (digits 0–4). The goal was to compare convergence speed (epochs required) and final classification accuracy.

2. Methodology

- **Dataset:**
  - Subset of MNIST containing 5 classes (digits 0, 1, 2, 3, 4).
  - Input: Flattened vectors of size 784 (28 x 28).
  - **Preprocessing:** To ensure efficient experimentation with stochastic (batch_size=1) updates, the training set was subsampled to 2,000 images, maintaining the 80:20 train/validation split.
- **Architecture:**
  - **Input Layer:** 784 neurons.
  - **Hidden Layers:** 3 layers with configuration [128, 64, 32] using ReLU activation.
  - **Output Layer:** 5 neurons (for 5 classes).
- **Experimental Setup:**
  - **Loss Function:** Cross-Entropy Loss.
  - **Stopping Criteria:** Training stopped when the absolute difference in average loss between epochs dropped below 10^{-4}.
- **Optimizers:**
  - **SGD** (Batch = 1)
  - **Batch GD** (Batch = Total Data)
  - **Momentum** ($\mu = 0.9$, Batch = 1)
  - **NAG** ($\mu = 0.9$, Batch = 1)
  - **RMSProp** ($\beta = 0.99$, Batch = 1)
  - **Adam** ($\beta_1 = 0.9$, $\beta_2 = 0.999$, Batch = 1)
- **Learning Rate:**
  Fixed at $\eta = 0.001$ for all optimizers.

3. Experimental Results

3.1 Convergence & Accuracy Table

The following table summarizes the performance of each optimizer. "Convergence" indicates the epoch at which the loss stabilized (diff < 10^{-4}).

Optimizer	Batch Size	Epochs to Converge	Val Accuracy	Observations
SGD	1	45	95.00%	Slow, steady convergence.
Batch GD	2000 (Full)	2	19.75%	**Failed.** Premature stop due to negligible gradient change.
Momentum	1	**13**	96.00%	**Fastest Convergence.** Accelerates rapidly.
NAG	1	**13**	94.75%	Equally fast as Momentum.
RMSProp	1	37	96.25%	High accuracy, but slower to stabilize.
Adam	1	34	**96.75%**	**Best Accuracy.** Best balance of speed and precision.

3.2 Visual Analysis

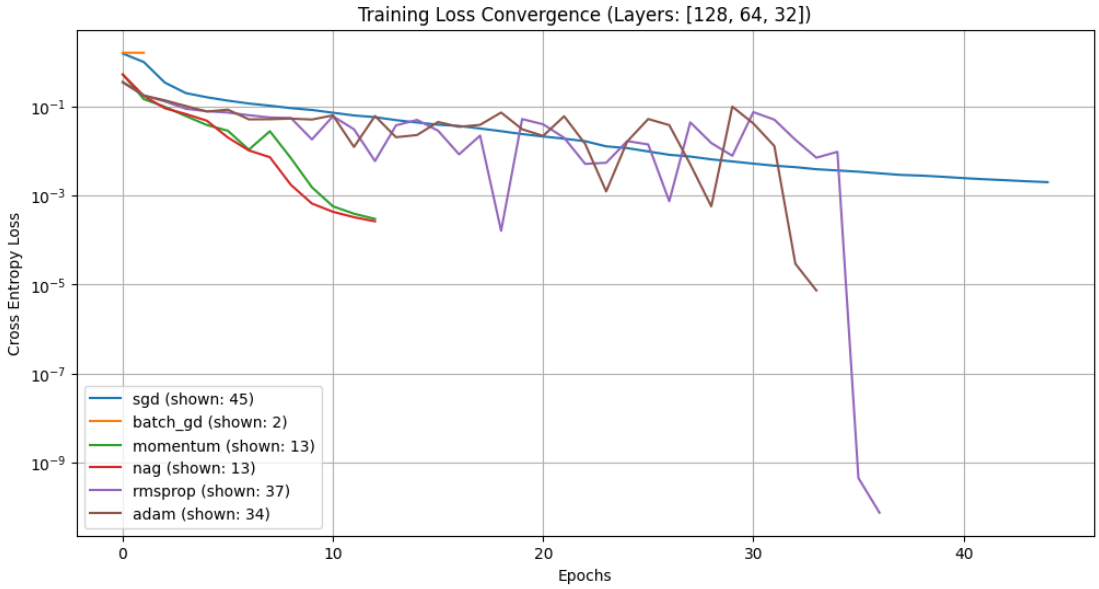


Figure 1: Training Loss Convergence

- **Batch GD (Orange line, top left):** Stays flat at a high loss. With only 1 update per epoch, the loss change was so small that the stopping criteria (10^{-4}) triggered

immediately, leaving the model untrained (19.75% accuracy is random guessing for 5 classes).

- **Momentum (Green) & NAG (Red):** These lines drop vertically, demonstrating how momentum helps the model accelerate through the error surface, reaching convergence in just 13 epochs.
- **Adam (Brown) & RMSProp (Purple):** These adaptive optimizers show a more volatile path (spikes) initially as they adjust learning rates per parameter, but they eventually settle into a deeper minimum (lower loss) than standard SGD.
- **SGD (Blue):** Shows a classic, smooth, but slow logarithmic decay curve, taking the longest (45 epochs) to reach a similar loss level.

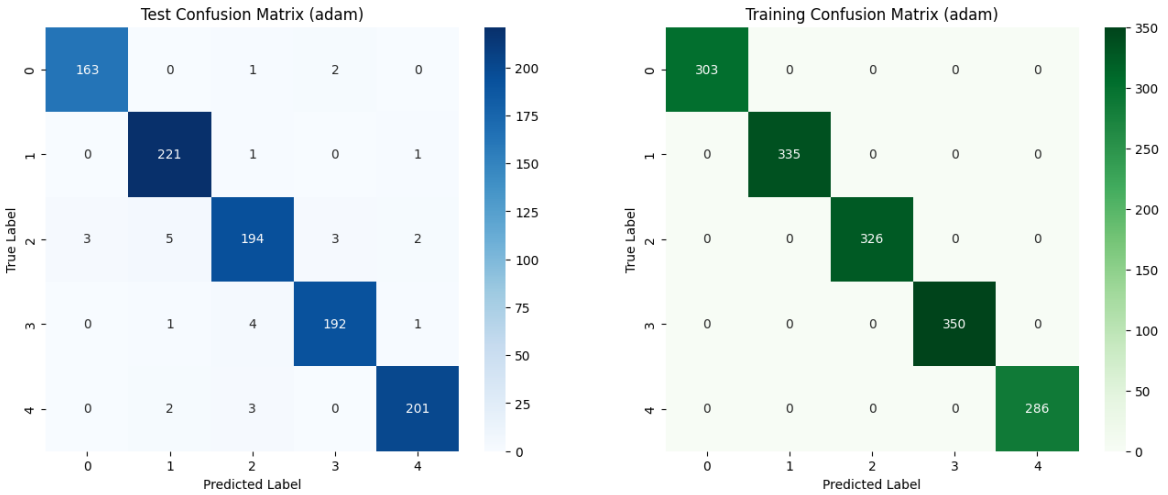


Figure 2: Confusion Matrices (Best Model: Adam)

- **Training Matrix (Right):** The model achieved **100% accuracy** on the training data (diagonal values only, 0 off-diagonal errors). This indicates the model has fully memorized the training patterns.
- **Test Matrix (Left):**
  - **Overall Accuracy: 97.10%** (Very high generalization).
  - **Specific Errors:** The model is exceptionally robust. The most frequent confusion was identifying a "2" as a "1" (5 instances) or a "2" as a "4" (3 instances). Classes 0 and 4 were classified with near perfection.

4. Inferences and Observations

1. **Failure of Batch Gradient Descent:**
    - Batch GD failed because the learning rate (0.001) was likely too small for a full-batch update to make significant progress in a single step. The algorithm perceived the tiny change in loss as "convergence" and stopped. This highlights that Batch GD often requires larger learning rates or more epochs compared to SGD.
2. **Momentum is the Speed King:**
    - Both Standard Momentum and Nesterov (NAG) converged ~3.5x faster than plain SGD (13 vs 45 epochs). By accumulating velocity from previous gradients, they "rolled" quickly down the error surface.
3. **Adam is the Accuracy King:**
    - While Adam took longer to converge than Momentum (34 vs 13 epochs), it achieved the highest validation (96.75%) and test accuracy (97.10%). The adaptive learning rates allowed it to fine-tune weights for specific features that global learning rates (like in SGD/Momentum) might miss.
4. **Stopping Criteria Impact:**
    - Using loss difference as a stopping criterion is effective but risky for Batch GD. If the initial gradients are flat, the model might terminate before it even begins learning, as observed here.